VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Nástroj pro testování uživatelského rozhraní webových aplikací

A Tool for Testing User Interface of Web-base Applications

2017 Radim Polach

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Radim Polach**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Nástroj pro testování uživatelského rozhraní webových aplikací
A Tool for Testing User Interface of Web-based Applications

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem práce je vytvořit komplexní nástroj pro testování uživatelského rozhraní webových aplikací. Nástroj bude dostupný formou rozšiřujícího modulu (plug-in) pro webový prohlížeč (např. Chromium browser). Uživateli umožní vytvářet testy formou nahrání scénáře (posloupnosti akcí) v aplikaci a jejich znovu spuštění. Pro každý scénář bude definován očekávaný stav, který se srovná vůči získanému výsledku. Důležité je, aby nahrané scénáře byly nezávislé na grafickém rozvržení (layoutu) stránky. Testovací prostředí umožní vytvořené testy hromadně spouštět s tím, že u neúspěšných testů uživateli nabídne sekvenci akcí vedoucí k chybě ve formě spustitelného skriptu. Výsledný nástroj bude využit k testování již značně komplexního uživatelského rozhraní webové aplikace určené pro plánování lidských zdrojů.

1. Seznamte se s metodami a nástroji určených pro testování softwaru. Zaměřte se na nástroje pro testování uživatelského rozhraní.
2. Navrhněte nástroj dle výše zmíněného popisu pro vytváření a spouštění testů.
3. Navržené řešení implementujte.
4. Připravte sadu testů prokazující funkčnost a použitelnost výsledného řešení.
5. Zhodnoťte možnosti navrženého řešení.

Seznam doporučené odborné literatury:

[1] Glenford J. Myers, Corey Sandler, Tom Badgett: The Art of Software Testing
[2] http://www.seleniumhq.org/

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Martin Šurkovský**

Datum zadání: 01.09.2016

Datum odevzdání: 14.07.2017

doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 14. července 2017 .....

Rád bych na tomto místě poděkoval Martinu Šurkovskému, za trpělivou práci se mnou a za užitečné rady, protože bez nich by tato práce nevznikla.

Abstrakt

Cílem této bakalářské práce je tvorba komplexního nástroje pro testování uživatelského rozhraní webových aplikací. Jeho forma bude odpovídat podobě doplňku pro prohlížeč Google Chrome. Testovací nástroje, kterých je možno používat nám dovolují otestovat v prostředí webu většinu *GUI* aplikací, avšak v dnešní době je mnoho komplexních webových aplikací vytvořeno v prvku *HTML5 Canvas*. Právě ten tvoří aspekt, jež nám dostupné nástroje nedovolují otestovat. Tato skutečnost je primárním motivem pro vytvoření nového nástroje, s jehož pomocí lze tyto aplikace otestovat.

Klíčová slova: Testování softwaru, Testování *GUI*, *Javascript*, *HTML5 Canvas*

Abstract

The target of this bachelor thesis is to create a complex tool for testing user interface of web applications. This tool will be available as a Google Chrome add-on. Available testing tools allow us in web test most GUI applications, but nowadays many complex web applications are created in *HTML5 Canvas*. It is this aspect of our available tools that we do not allow to test. This is the main motive for creating a new tool to test these applications.

Key Words: Software testing, Testing of *GUI*, *Javascript*, *HTML5 Canvas*

Obsah

Definice použitých zkratek a pojmů

HTML5 Canvas	–	Část specifikace *HTML5*, jež definuje kreslící plátno
Content script	–	Část doplňku prohlížeče Google Chrome, která se stará o přímou komunikaci s webovou aplikací
Background page	–	Část doplňku prohlížeče Google Chrome, starající se o komunikaci mezi částmi tohoto rozšíření
Devtools page	–	Část doplňku prohlížeče Google Chrome, která se stará o rozšíření vývojářských nástrojů
MVC	–	Architektura softwaru, sestavající se ze tří komponent (*Model*, *View* a *Controller*)
RIA	–	Rich Internet Application jsou webové aplikace založené na principu interakce uživatele s grafickým rozhraním

Seznam obrázků

Seznam tabulek

výpisů zdrojového kódu

1 Úvod

Grafické uživatelské rozhraní, dále jen *GUI*, představuje z pohledu uživatele formu, s jejíž pomocí interaguje s danou aplikací. Uživatel zadává jednotlivé úkony, které znamenají vstupy do aplikace a na jejich základě očekává určitou reakci, tudíž výstupy aplikace. *GUI* tedy reprezentují obrazovky na nichž se uživateli zobrazují jednotlivé prvky tohoto rozhraní jako například formuláře, tlačítka a další grafické prvky. *GUI* představuje v architektuře *MVC* komponentu *View*, která se stará o zobrazení dat a také *Controller*, který reaguje na podněty uživatele.

Testování *GUI* rozhraní je aspekt testování softwaru, pomocí nějž testujeme interakci mezi uživatelem a aplikací. Výstupy aplikace ověřujeme na základě vstupních dat, zadaných uživatelem. Také testujeme validaci vstupních polí formulářů. Zjišťujeme, zdali jsou vyplněná všechna povinná data a jestli jejich formát odpovídá požadovanému formátování. Řešíme také konzistentnost prvků formuláře, zda se zobrazí všechny prvky formuláře ve správném počtu a zda jsou tyto prvky znázorněny ve správném pořadí. Dále testujeme, jak jsou prvky *GUI* na dané obrazovce umístěny. Zobrazují-li se na správných pozicích a mají-li odpovídající grafický formát. Ať už barvu, velikost písma, či tvar. Poslední částí testování *GUI* je ověřování funkčnosti jednotlivých prvků *GUI*. Testujeme odeslání formuláře s daty na zpracování správnou funkcí, nebo zda odkaz přesměruje uživatele na správný web.

Dostupné nástroje nám dovolují otestovat drtivou většinu aspektů. Můžeme zmínit velmi populární nástroj Selenium [1], např. v podobě doplňku pro prohlížeč Mozzila Firefox. Existuje ovšem aspekt, který tento nástroj opomíjí. Tím je prvek kreslící plátno *HTML5 Canvas* [2]. S dostupnými nástroji jsme schopni otestovat jeho umístění na obrazovce, či kontrolovat události myší nad tímto prvkem, ale nelze jakkoli otestovat prvky, vykreslené uvnitř, či jejich chování. Tato skutečnost je primární motivací pro vznik nového nástroje, který bude moci tento opomíjený aspekt otestovat. Nástroj je vyvíjen v podobě doplňku pro prohlížeč Google Chrome. Tento nástroj by měl být schopen otestovat komplexní aplikaci napsanou ve výše zmíněném prvku.

Cílem této bakalářské práce je vytvoření testovacího nástroje ve formě doplňku prohlížeče Google Chrome. Nástroj se bude nacházet v panelu vývojářských nástrojů. Podpora testování tímto nástrojem bude zajištěna rozhraním, které bude testovaná webová aplikace implementovat. Nástroj umožní tvorbu testovacích scénářů a jejich opětovné spouštění.

[1] urlhttp://seleniumhq.org/

[2] `https://www.w3schools.com/html/html5_canvas.asp`

2 Analýza požadavků

Testovaná aplikace i tento nástroj budou používány v rámci prohlížeče Google Chrome, který pro nás bude představovat primární doménu. Aplikace pro testování bude dostupná ve formě doplňku pro tento prohlížeče. Vytvořený doplněk bude sloužit jako nástroj pro testování grafického uživatelského rozhraní webových aplikací. Uživateli umožní testovat aplikaci, vytvořenou v rámci *HTML5 Canvas*. Tohoto uživatele bude představovat vývojář testované aplikace. Danou aplikaci bude možné ve vytvořeném nástroji otestovat pomocí vytvořených testovacích scénářů. Každý test bude mít očekávaný stav, jež se srovná vůči získanému výsledku. Testy bude možné uložit v aplikaci pro možnost opětovného spuštění. Testy budou nezávislé na layotu neboli rozložení stránky. Proto rozdílné rozložení stránky neovlivní opětovné spuštění testů. Tyto testy bude možno dále hromadně spouštět, a v případě neúspěšného testu nástroj nabídne spustitelný skript se sekvencí vedoucí k chybě.

2.1 Funkční požadavky

Výsledkem analýzy jsou tyto funkční požadavky. Na obrázku 1. si popíšeme případy užití doplňku prohlížeče Google Chrome. V tabulkách 1., 2., 3. vidíme podrodný popis jednotlivých případů užití.

Získání reference primárního Javascriptového objektu webové aplikace Nástroj nám umožní získávat referenci na objekt kreslícího plátna, který pod sebou zastřešuje objekty a jejich chování. S touto referencí budeme moci přistoupit k rozhraní aplikace. Skrze ní lze pracovat se všemi dostupnými metodami a objekty.

Získání Javascriptové reference objektu kreslícího plátna Nástroj nám umožní získat referenci na objekt, nacházející se uvnitř kreslícího plátna. Bude možné získat nebo změnit jeho stav. Nakonec bude možné takto získané objekty porovnávat, či vytvořit kopii takovéhoto objektu.

Získání offsetu objektu kreslícího plátna V nástroji budeme moci dále nejen pracovat se získanými objekty, ale budeme moci získávat souřadnice objektu, vzhledem k pozici kreslícího plátna. Pomocí těchto souřadnic budeme moci při tvorbě scénáře získat voláním odpovídající metody na primární referenci objekty, které jsou na těchto souřadnicích umístěny.

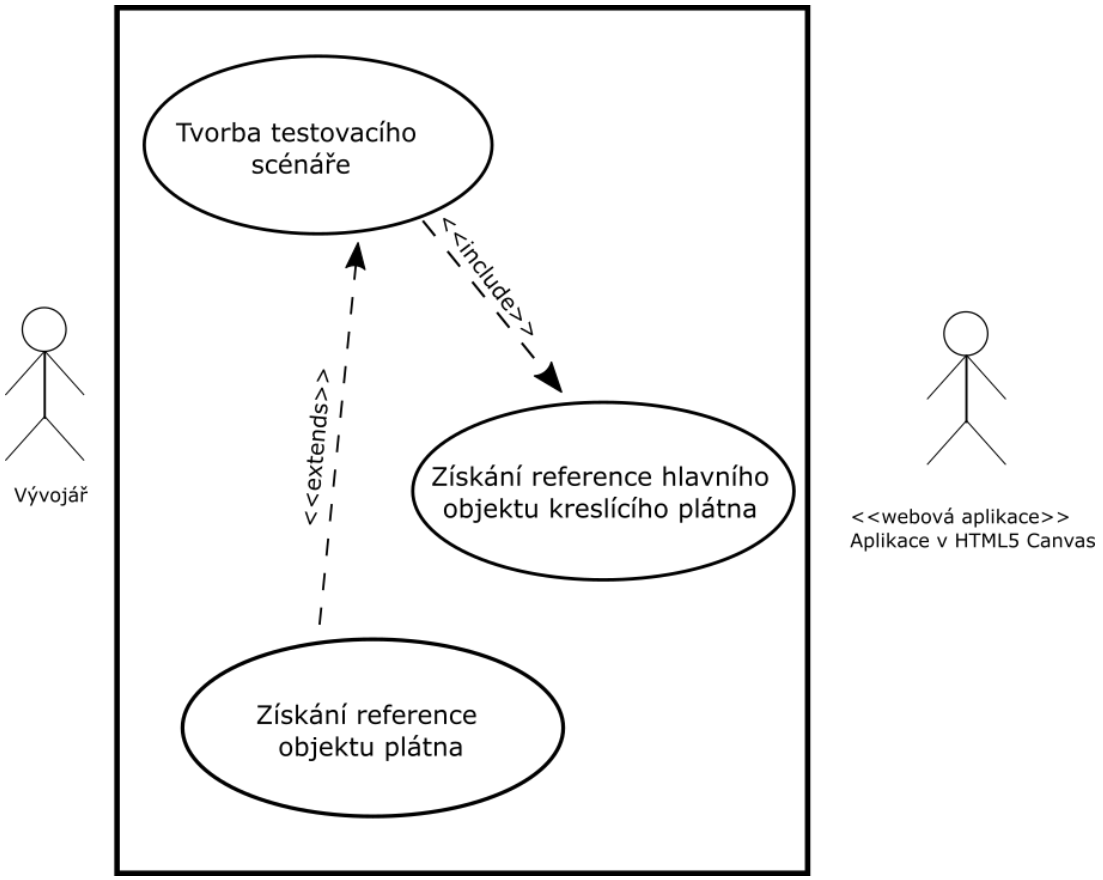
Výběr objektu kreslícího plátna Nástroj nám umožní vybírat objekty v testované webové aplikaci. Výběr bude realizován v podobě návrhového vzoru *Event-Listener* [6], a ten bude naslouchat nad událostmi kliknutí myši nad kreslícím plátnem.

2.2 Nefunkční požadavky

Implementace testovacího *API* webovou aplikací Testovaná webová aplikace musí implementovat rozhraní pro testování *GUI*. K tomuto rozhraní budeme moci přistupovat při tvorbě scénáře. Toto rozhraní bude obsahovat metody, které budou zprostředkovávat potřebnou funkcionalitu, umožnující testování webové aplikace.

Nástroj bude ve formě doplňku pro prohlížeč Google Chrome Nástroj pro testování bude vyvíjen ve formě doplňku pro prohlížeč Google Chrome. Nástroj bude muset splňovat specifikace a formu takovéhoto doplňku.

Inicializace a notifikace nástroje Při spuštění nástroje dojde k získání primárního referenčního objektu. Pokud dojde při události kliku myší k získání objektu, ten se uloží a nástroj pro testování bude o události získání objektu notifikován.



Obrázek 1: Jak aktéři reagují s jednotlivými aktivitami

Tabulka 1: Use case «Tvorba testovacího scénáře»

Aktéři	• Vývojář
Podmínky spuštění	Nástroj pro testování musí být spuštěn
Hlavní průběh	1. Vývojář stiskne tlačítko pro tvorbu scénáře. 2. Nástroj zobrazí pole pro název scénáře. 3. Vývojář vyplní název scénáře. 4. Vývojář stiskne tlačítko pro uložení testovacího scénáře. 5. Nástroj nově vytvořený scénář označí jako vybraný. 6. Nástroj uloží testovací scénář.
Podmínky dokončení	Testovací scénář je uložen
Alternativní průběh	1. Vývojář omylem nevyplní název 2. Nástroj zobrazí upozornění

Tabulka 2: Use case «Získání reference primárního objektu plátna»

Aktéři	• Vývojář • Webová aplikace
Podmínky spuštění	V nástroji pro testování je vytvořen scénář
Hlavní průběh	1. Nástroj pro testování vyvolá ve webové aplikaci požadavek na referenci primárního objektu kreslícího plátna. 2. Webová aplikace obdrží požadavek o referenci primárního objektu kreslícího plátna. 3. Reference primárního objektu je uložena do paměti. 4. Testovací nástroj je notifikován o uložení primárního objektu kreslícího plátna do paměti. 5. Testovací nástroj vloží jako první příkaz referenci do paměti na primární objekt. 6. Příkaz se zobrazí vývojáři jako první příkaz testovacího scénáře.
Podmínky dokončení	Prvním příkazem vytvořeného testovacího scénáře je primární objekt kreslícího plátna

Tabulka 3: Use case «Získání reference objektu plátna»

Aktéři	• Vývojář • Webová aplikace
Podmínky spuštění	V nástroji pro testování je vytvořen scénář
Hlavní průběh	1. Vývojář stiskne tlačítko pro vybrání objektu kreslícího plátna. 2. Vývojář klikne myší na objektu uvnitř kreslícího plátna. 3. Nástroj pro testování vyvolá ve webové aplikaci požadavek na referenci objektu kreslícího plátna na daných souřadnících kliknutí myši. 4. Webová aplikace obdrží požadavek o referenci objektu kreslícího plátna. 5. Reference objektu je uložena do paměti. 6. Testovací nástroj je notifikován o uložení objektu kreslícího plátna do paměti. 7. Testovací nástroj vloží příkaz jako referenci do paměti na získaný objekt. 8. Příkaz se zobrazí vývojáři jako další příkaz testovacího scénáře.
Podmínky dokončení	Příkazem vytvořeného testovacího scénáře je objekt kreslícího plátna

3 Aktuální stav

Shrneme si metodiku testování softwaru. Zaměříme se na nástroje pro testování uživatelského rozhraní. Dozvíme se jejich možnost použití a nakonec si řekneme něco i o aspektu, který s nimi nelze otestovat.

3.1 Testování softwaru

Testování softwaru je proces, jež se uplaťnuje při různých etapách vývoje softwaru. Tento pojem lze rozdělit na dva základní principy. Prvním druhem je testování *black box* [1]. Toto testování lze provádět již na základě specifikace softwaru. Mezi tento druh testování se řadí statické testování [5], jelikož v ruce nemáme žádný funkční softwaru, takže testujeme něco, co neběží. Opačným případem je dynamické testování. Toto testování lze uplatnit v případě, že existuje prototyp softwaru. V této etapě vývoje softwaru testujeme aplikaci pomocí vstupů a na základě jejich stavů očekáváme určité výstupy. V této částí využíváme metodu rozdělení vstupů na třídy ekvivalence, které nám umožnují snížit množinu vstupů na zvládnutelný počet a hraniční podmínky, kterými testujeme chování softwaru při nestandartních vstupech. Ani v jednom z uvedných případů tedy nevíme nic o chování vnitřní struktury programu. Přistupujeme tedy k softwaru z uživatelského pohledu.

Dalším pricipem je testování *white box* [1], mezi které lze zařadit *unit testy*, *integrační testy* [2] a v poslední řadě *testy akceptační*. V této fázi zkoumáme strukturu programu zevnitř, tedy samotný zdrojový kód. Těmi nejzákladnějšími testy jsou *unit testy*, které testují program na úrovní samotných tříd či modulů a zkoumají, zdali nově přidaná nebo změněná funkcionalita nezmění korektní chování programu. Složitější testování představují integrační testy. Tyto testy testují propojení komponent na úrovni jednotlivých modulů nebo aplikací do jednoho funkčního celku. Testujeme tedy vzájemnou komunikaci a propojitelnost těchto komponent mezi sebou. Akceptáční testy jsou nejdůležitější pro dodání aplikace z pohledu zákazníka. Provádí se vetšinou právě u zákazník na zakladě předpřipravených scénářů.

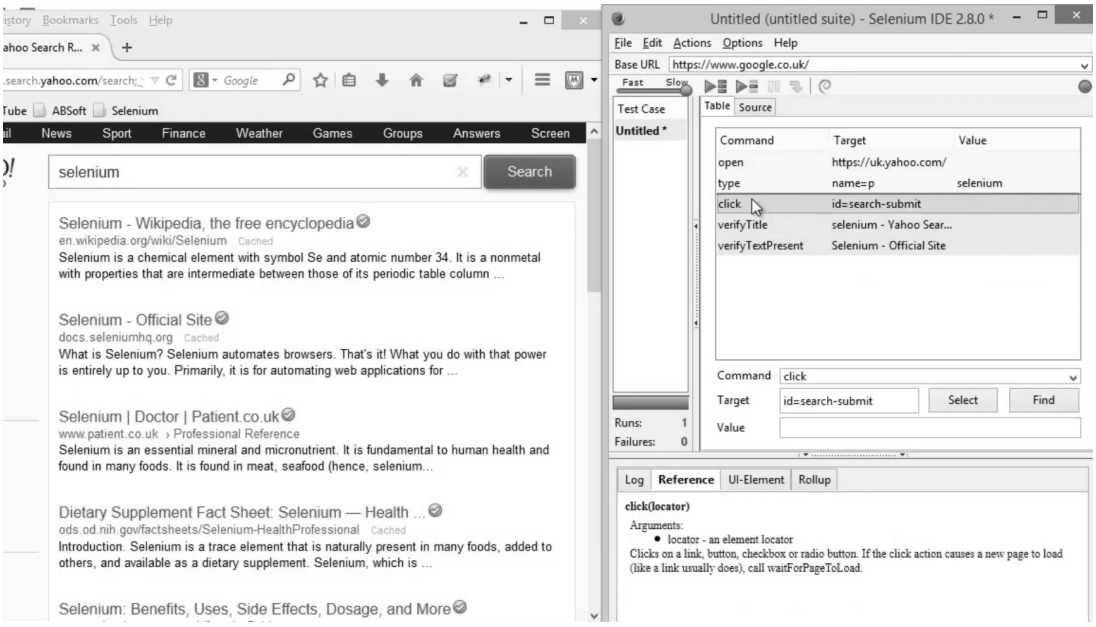
3.2 Nástroje pro testování uživatelského rozhraní

Na dnešním poli existuje několik nástrojů pro testování nebo pro automatizaci úkonů s tím spojených. Cílem nástrojů pro testování uživatelského rozhraní je ověření koreknosti chování aplikace. V případě grafického uživatelského rozhraní webových stránek se jedná o nástroje založené na několika rozdílných principech. Prvním principem je vyžití testovacích frameworků, napsaných v Javascriptovém jazyce. V tomto případě jsou testy spuštěny přímo v prostředí webové stránky. Jiný přístup tkví v získávání jednotlivých stavů aplikace v podobě obrazovek. Při provádění takovýchto testů se kontroluje samotný layout webové stránky v jednotlivých stavech aplikace. Poslední metodou je komplexnější přístup a kontrola přímého chování webové stránky. Tento princip spočívá v kontrole elementů samotného *DOMu* stránky a také jeho událostem, jako

např. kliknutí, odkázaní na hypertextový odkaz menu, či kliknutí na tlačítko pro přihlášení do aplikace. Při spouštění těchto testů se tedy v jednotlivých krocích provádí dříve zmíněné uživatelské akce a kontrolují samotnou konfiguraci aplikace. Příkladem chybné konfigurace mohou být chybějící nebo nevalidní data ve formuláři. Poslední přístup k testování je z výše zmíněných nejefektivnější.

3.3 Selenium IDE

Jedním z nejvýznamějších nástrojů je komplexní nástroj Selenium [3]. Je určen pro automatizaci testování webových aplikací, která umožní zkrátit čas testování. Tento software existuje v několika variantách. Selenium vytváří testy postupným nahráváním jednotlivých kroků. Uživatelsky nejpřívětivější varianta je v podobě doplňku pro prohlížeč Mozzila Firefox. Tento nástroj je napsán v Javascriptu. Nejjednodušší cesta k použití je instalace doplňku prohlížeče Mozzila Firefox. Toto jednoduché řešení ovšem přináší omezení v podobě dostupné platformy webového prohlížeče, podobně jako vyvíjený nástroj pro tesování uživatelského rozhraní webových aplikací. Užití tohoto nástroje spočívá v záznamu jednotlivých úkonů uživatele v prohlížeči. V tomto bodě Selenium IDE nahrává jakoukoli akci. V průběhu nahrávání se zaznamená např. kliknutí na odkaz v menu nebo kliknutí na obrázek, či vyplnění údaje do formuláře. V průběhu nahrávání ale můžeme tyto příkazy i ručně vložit. Po ukončení nahrávání se scénář uloží do seznamu nahraných scénářů, ze kterého jej lze spouštět. Podobu tohoto doplňku a jeho rozhraní můžeme vidět na obrázku 2.

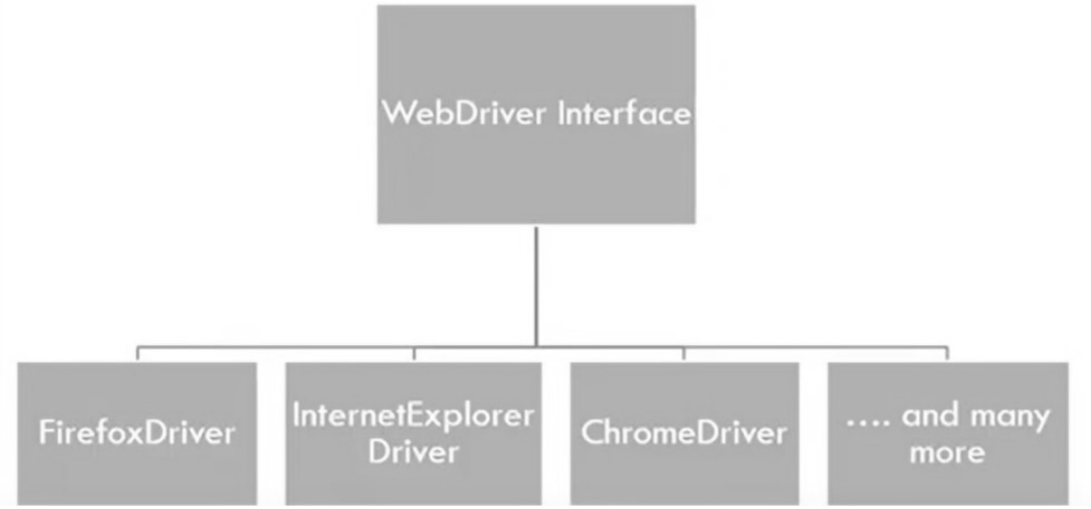


Obrázek 2: Selenium IDE

[3] http://www.seleniumhq.org/

3.4 Web Driver

Verze Selenia, zvaná Web Driver řeší výše zmíněný problém omezení na jedinou platformu. Jedná se o kompletní programově orietované *API* pro efektivní řízení prohlížeče. Řeší také podporu moderních dynamických webových stránek. Podstata těchto stránek tkví v tom, že se obsah dynamicky mění v různých částech bez toho, aby byla znovu načtena. Poskytuje také lepší podporů jednotlivým, nově vzniklým paradigmatům při testování moderních webových aplikací. Toto *API* využívá volání nativních metod prohlížeče pro jeho řízení. Obsahuje implementaci pro Safari, Google Chrome a mnoho dalších prohlížečů. Rozsah použití se tedy oproti verzi Selenium IDE pro prohlížeč Mozzila Firefox ohromně rozšiřuje. Tyto testy se píší v programovacích jazycích *Javascript*, *Java*, *C#* atd. Pro tvorbu testů pomocí tohoto nástroje je ještě třeba použít podle vybraného programovacího jazyka odpovídající klientský ovladač. Schéma tohoto *API* je přehledně zobrazeno na obrázku 3.



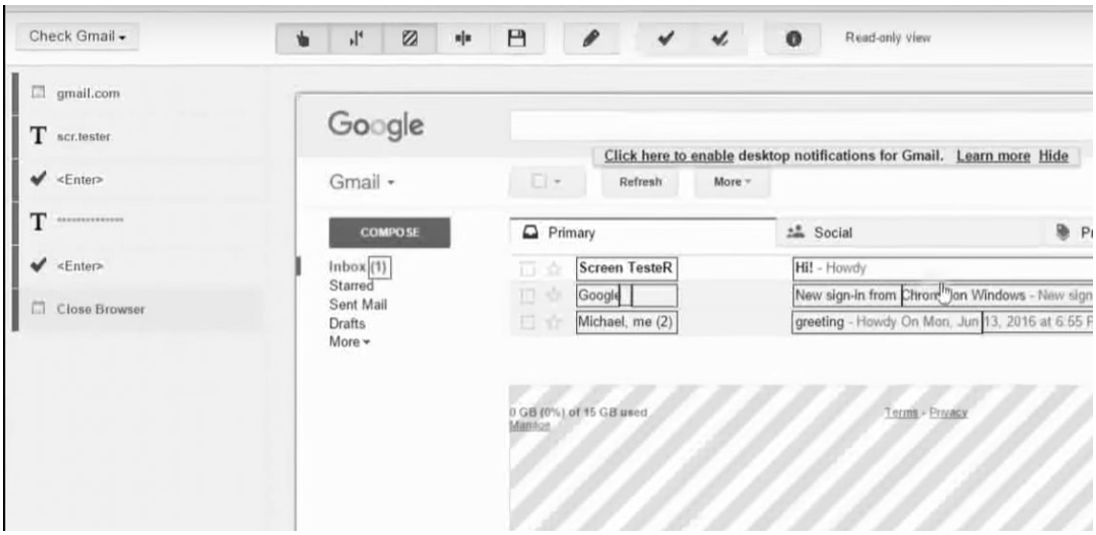
Obrázek 3: Web Driver Interface

3.5 Alternativní nástoje

Z možných alternativ Selenia je možné zmínit Screenster [4]. Tento nástroj je dostupný ve formě webové aplikace. Tak jako Selenium nahrává jednotlivé kroky do scénáře. Liší se ale přístupem v tom, že pro každý krok zaznamenává snímek obrazovky a ukládá jej jako výchozí. U jednotlivých kroků lze definovat dynamické části obrazovky, které nebudou porovnávány. Screenster dále tyto kroky uloží do testu, aby jej bylo možné opětovně spustit. Při opětovném spuštění prochází jednotlivé obrazovky a sebemenší rozdíly jsou na obrazovkách zvýrazněny. Podobu uživatelského rozhraní ScreenSteru je možno uvidět na obrázku 4. Jednou z posledních možností je využití testovacího frameworku Dojo Toolkit [5]napsaného v programovacím

[4] `http://www.screenster.io/`
[5] `https://dojotoolkit.org/`

jazyce *Javascript*. Obsahuje kolekci nejčastěji používaných *API* a knihovnu, obsahující celou řadu funkcí. Můžeme jej integrovat do stránky z knihoven *Google Ajax API* a jednoduše začít testovat.



Obrázek 4: Screenster *GUI*

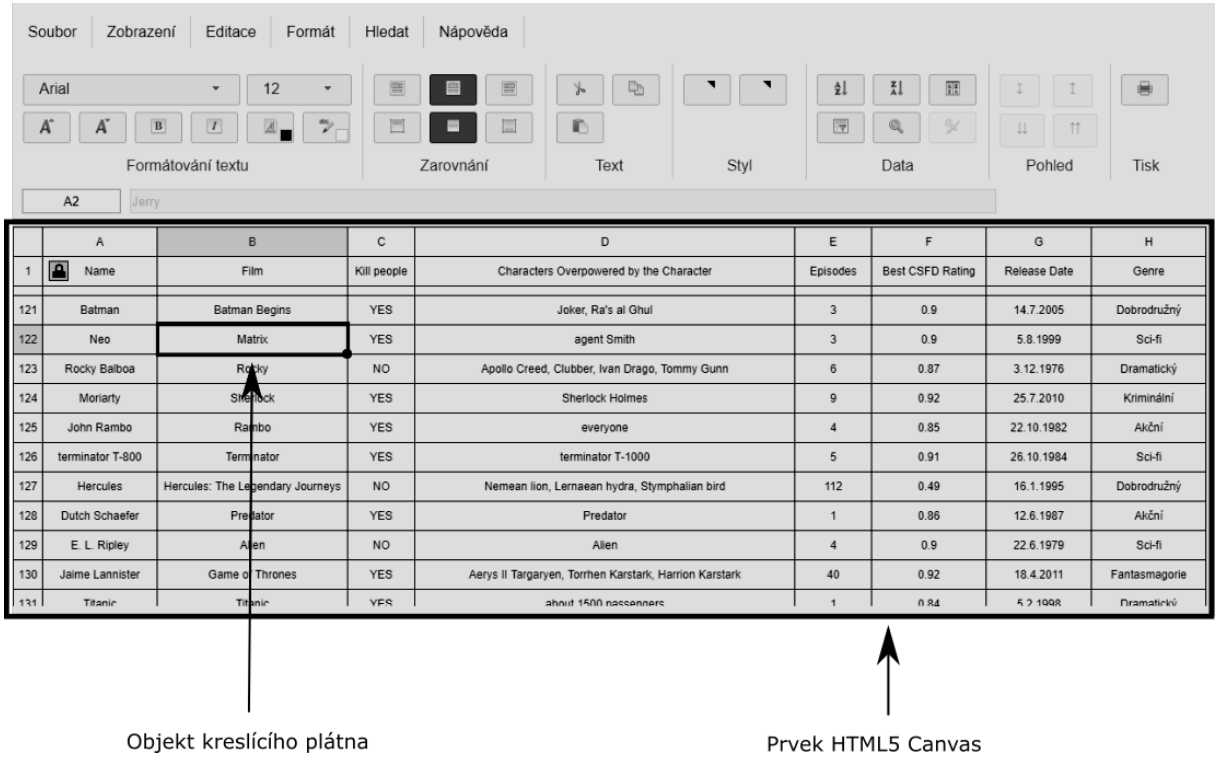
3.6 Rich Internet Aplication

Zkráceně *RIA* jsou moderní a komplexní webové aplikace na principu ovládání pomocí přímé interakce uživatele s grafickým rozhraním, podobně jako klasické desktopové aplikace. Dalším primárním rysem je obnovování jen určité části stránky. Tyto části poté nazýváme *živé oblasti*, protože jejich obsah se dynamicky mění. Tyto aplikace jsou často napsány v prvku *HTML5 Canvas*. Kreslící plátno tvoří prostředí, do něhož je aplikace schopna vizualizovat svůj stav a umožňuje uživateli skrze toto rozhraní s aplikací pracovat, ovládat ji. Příkladem takové webové aplikace je poštovní klient nebo webová hra.

Problémem výše zmíněných nástrojů z pohledu testování těchto aplikací je, že nám dovolují přistoupit k *HTML5 Canvas* jen jako k jinému elementu. Selenium nám dokáže otestovat, zdali jsme *HTML5 Canvas* element označili například myší. Nedokáže nám však detekovat objekty uvnitř kreslícího plátna. Můžeme tedy ověřit např. jeho nastylování nebo rozměry. Můžeme také zkontrolovat, či nastavit jeho atributy nebo umístění na stránce. Pomocí nástroje Screensteru zase můžeme zkontrolovat i jeho aktuální stav, ale jen jako část obrazovky, nikoli samotné objekty. Věc jako získání objektu z kreslícího plátna je všem těmto nástrojům nepřístupná. Tímto se dostáváme do pozice vývojáře, který vyvíjí *RIA* webovou aplikaci s využitím prvku *HTML5 Canvas*, avšak nemá možnost testovat tuto aplikaci již zmíněnými nástroji. Cílem této bakalářské práce je vytvořit nástroj, s jehož pomocí jsem schopni tyto aplikace otestovat.

Příklad *RIA* aplikace vidíme na obrázku 5. Taková aplikace tedy na první pohled vypadá jako normální webová stránka. Po bližším ohledání však vidíme, že rozsáhlou část této stránky

zabírá právě prvek *HTML5 Canvas*. Menu pro ovládání je umístěno nad ním. Toto menu tedy jsme schopni např. nástrojem Seleniem otestovat. Jakmile však příjde řada na *HTML5 Canvas*, nezjístíme nic o jeho vnitřním stavu. V tomto kokretním případě se jedná o tabulku se záznamy. My nemůžeme zjístit ani obsah jediné buňky, protože ta je jako objekt kreslícího plátna pro Selenium neviditelná.



	A	B	C	D	E	F	G	H
1	Name	Film	Kill people	Characters Overpowered by the Character	Episodes	Best CSFD Rating	Release Date	Genre
121	Batman	Batman Begins	YES	Joker, Ra's al Ghul	3	0.9	14.7.2005	Dobrodružný
122	Neo	Matrix	YES	agent Smith	3	0.9	5.8.1999	Sci-fi
123	Rocky Balboa	Rocky	NO	Apollo Creed, Clubber, Ivan Drago, Tommy Gunn	6	0.87	3.12.1976	Dramatický
124	Moriarty	Sherlock	YES	Sherlock Holmes	9	0.92	25.7.2010	Kriminální
125	John Rambo	Rambo	YES	everyone	4	0.85	22.10.1982	Akční
126	terminator T-800	Terminator	YES	terminator T-1000	5	0.91	26.10.1984	Sci-fi
127	Hercules	Hercules: The Legendary Journeys	NO	Nemean lion, Lernaean hydra, Stymphalian bird	112	0.49	16.1.1995	Dobrodružný
128	Dutch Schaefer	Predator	YES	Predator	1	0.86	12.6.1987	Akční
129	E. L. Ripley	Alien	NO	Alien	4	0.9	22.6.1979	Sci-fi
130	Jaime Lannister	Game of Thrones	YES	Aerys II Targaryen, Torrhen Karstark, Harrion Karstark	40	0.92	18.4.2011	Fantasmagorie
131	Titanic	Titanic	YES	about 1500 passengers	1	0.84	5.2.1998	Dramatický

Obrázek 5: *RIA* aplikace pro plánování lidských zdrojů

4 Návrh řešení

V této kapitole se budeme zabývat návrhem nástroje pro testování grafického uživatelského rozhraní webových aplikací. Vysvětlíme si, co musí splňovat aplikace, aby ji bylo možné pomocí tohoto nástroje testovat. Ukážeme si, jak má vypadat nástroj, aby byl doplňkem pro prohlížeč Google Chrome.

4.1 Google Chrome

Základní doménou, pro kterou bude nástroj vyvíjen je freeway prohlížeč Google Chrome. Staví na otevřeném renderovacím jádře *Webkit*, jež byl od verze 28 nahrazen *Blinkem*. Lze jej nainstalovat na platformách *Windows*, *Linux*, *Apple* a *Android*. Existuje ve dvou verzích. První je určena pro 32 bitové operační systémy a druhá pro 64 bitové operační systémy. Integruje v sobě služby společnosti *Google* a synchronizaci *Google* účtu. Od verze číslo 9 lze využívat internetového obchodu *Chrome*, odkud lze stáhnout nové motivy, hry a v poslední řadě pro nás nejdůležitější doplňky. Nejnovější verze má pořadové číslo 59 [6].

4.2 Použité technologie

Vyspecifikujeme, jaké jazyky, pluginy a frameworky budeme při tvorbě nástroje používat. Definujeme si jejich význam a použití.

4.2.1 Javascript

Javascript je multiplatformní, interpretový a objektově orientovaný programovací jazyk. Jeho autorem je Brenda Eich [4]. Objektově orietovaného paradigmatu dosahuje pomocí *prototypování* [7]. Je nejvíce znám pro své užití jako skriptovací jazyk webových stránek, kdy běží na klientské straně. *Javascript* je ale možné využít i na straně serveru, a to pomocí implementace Node.js [8] nebo Apache. Standardem pro *Javascript* je *ECMA* (European Computer Manufacturers Association) [9].

4.2.2 jQuery

jQuery [3] znamená rozšíření Javascriptu pro lepší interakce mezi ním a *HTML*. Základní filozofií je oddělení chování od struktury. Představuje abstraktní vrstvu nad Javascriptem, zajištující kompatibilitu aplikace mezi různými prohlížeči a jejich různými vrstvami. Hlavními rysy je práce se *selektory* [10], podobně jako v *CSS* a intenzivní obsluha událostí.

[6] 14. července 2017
[7] `https://www.zdrojak.cz/clanky/oop-v-javascriptu-i/`
[8] `https://nodejs.org/en/`
[9] `https://www.ecma-international.org/`
[10] `https://api.jquery.com/category/selectors/`

4.2.3 Bootstrap

Je frontend webový framework pro tvorbu responzivních webových aplikací. Obsahuje sadu šablon, založených na *CSS*, sloužících pro úpravu typografie, tvorbu formulářů a navigace. Součástí jsou také interaktivní prvky jako tlačítka, boxy, či menu. Výhodou Bootstrapu [11] je jednoduché vytvoření jakéhokoli uživatelské rozhraní.

4.3 Testovací API

Toto rozhraní přestavuje univerzální prostředek, který splňuje požadavky pro testování *GUI* webových aplikací. Popis tohoto rozhraní a jeho metod nalezneme níže.

getObjectAtOffset(x, y):object metoda navrací objekt na daném offsetu vzhledem ke kreslícímu plátnu, vstupem jsou číselné souřadnice [x,y]

getObjectId():object metoda navrací unikátní identifikátor objektu

getObjectById(object):object metoda vrací objekt kreslícího plátna, vstupem je unikátní identifikátor objektu

getBoundingObjectRect():object metoda navrací souřadnice objektu vzhledem ke kreslícímu plátnu

getProperties():object metoda vrací seznam vlastností objektu

getProperty(string):object metoda navrací požadovanou vlastnost objektu kreslícího plátna, vstupem je název property

setProperty(string, object) metoda nastavuje konkrétní vlastnost kreslícího plátna, vstupem je název property a její nová hodnota

4.3.1 Testovaná aplikace

Aby bylo možné testovat webovou aplikace s naším nástrojem, bude třeba specifikovat minimální *API*, které tato aplikace bude muset implementovat. Zakladní myšlenkou pro toto *API* je, že kreslící plátno pro nás tvoří prostředí. Poté je pro nás důležité zjistit stav tohoto prostředí a také jeho jednotlivých prvků, jakožto prvky samotné, které pro nás představují určité objekty. Zakladní funkcionalita rozhraní tedy tkví v získání primárního objektu kreslícího plátna, který zastřešuje samotné prostředí aplikace.

[11] `http://www.getbootstrap.com/`

4.3.2 Objekty

Dalším krokem by měla být možnost získat jednotlivé prvky, tedy objekty uvnitř kreslícího plátna. Pokud jsme schopni takové prvky získat, bude určitě potřeba je rozlišovat. Z tohoto důvodu je nutné získat od každého objektu jeho identifikátor, který je pro něj unikátní v rámci celé aplikace. Samozřejmě nejen samotný primární objekt ale i jednotlivé jeho objekty mohou mít určitý stav, který se může měnit. Proto bychom v rámci tohoto rozhraní měli být schopni nejen tento stav získat, ale také jej přímo změnit.

4.3.3 Rozšíření

Krom těchto standardních metod by toto rozhraní mohlo mít i další metody, které by znamenaly funkcionalitu navíc a testovaná aplikace by je nutně nemusela implementovat. Příkladem takovéto metody by mohlo být získání statusu určitého prvku kreslícího plátna. V rámci specifikace tohoto rozhraní předpokládáme, že získávané objekty budou obdelníkového tvaru a testovaná aplikace bude obsahovat jediný prvek *HTML5 Canvas*.

4.4 Struktura doplňku

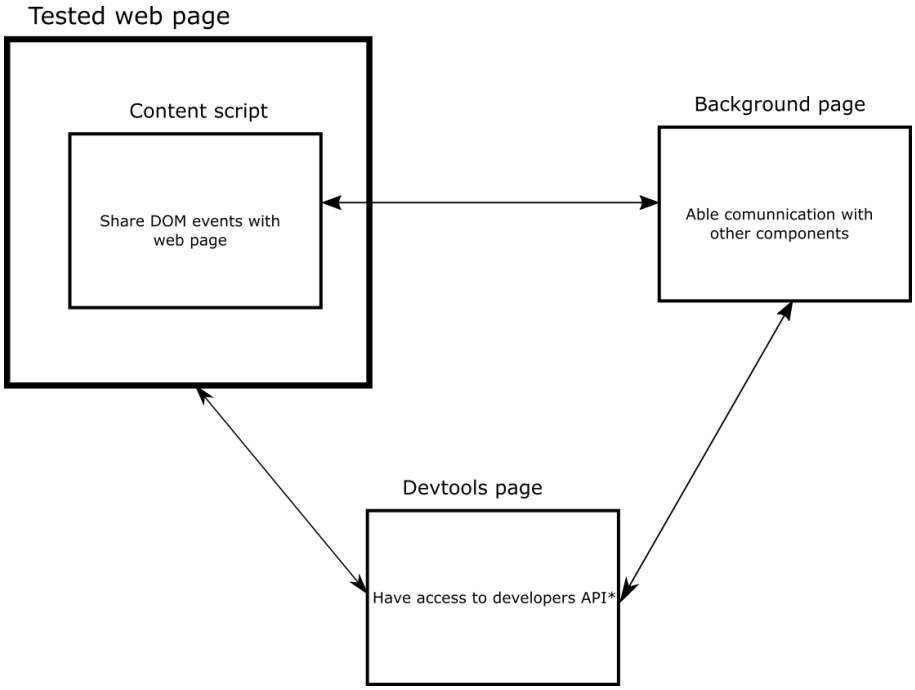
Vytvořený testovací nástroj bude mít formu doplňku prohlížeče Google Chrome. Doplněk lze v jádru označit jako klasickou webovou aplikaci, oproti které bude moci přistupovat k jednotlivým *API* platformy *Chrome*, jejichž funkcionalitu využijeme při vývoji nástroje. Podle obecné specifikace se takováto aplikace skládá z několika komponent (části). První část se nazývá *Content script* a zajišťuje přímou interakci s webovou stránkou a následnou komunikace s komponentou *Background page*. *Background page* vytváří komunikační spojení mezi *Content script* a *Devtools page*. *Devtools page* bude vytvářet poslední část naší aplikace, která zodpovídá za správu a provádění testů. Diagram komponent rozšíření nalezneme na obrázku 6.

4.4.1 Content script

Zakladní funkcí *Content script* je manipulace s *DOM* objekty webové stránky. Dále dokáže vyvolat a také zachytávat události na stránce, opět skrze *DOM*. Webová aplikace i *Content script* mají každá vlastní běhové prostředí Javascriptu. *Content script* s webovou stránkou tedy nesdílí objekt `window`, tudiž nemá ani přístup ke globálním proměnným stránky. Další funkčnost, kterou *Content script* zaštiťuje je spojení s komponentou *Background page* doplňku. Vytváří tedy takové rozhraní mezi webovou aplikací a doplňkem.

4.4.2 Background script

Background page má dvě možné podoby. Nejprve v podobě jedné statické stránky pro celé rozšíření, která běží celou dobu, po kterou je doplněk aktivní. Druhou a poslední podobou je dynamická stránka, zvaná *Event page*, která chováním připomíná *Content script*, takže každá



Obrázek 6: Komponentí diagram rozšíření

webová aplikace má vlastní *Event page*. Hlavní funkcí této komponenty je práce s webovým prohlížečem. Dokáže zachytit uživatelské akce vyvolané v prohlížeči, jako například otevření nové záložky nebo navštívení konkrétní stránky, ale dokáže je také vyvolat (automaticky otevře danou stránku v nové záložce prohlížeče). Další příkladem použití je možnost dynamického vkládání *Content scriptu* do webové stránky. Poslední funkcí je komunikace mezi jednotlivými komponentami rozšíření. Na straně jedné dokáže komunikovat s *Content scriptem*, a na straně druhé komunikuje s komponentou *Devtools page*.

4.4.3 Devtools page

Devtools page je komponenta, tvořící poslední část doplňku. Její primární funkcí je rozšíření vývojářských nástrojů prohlížeče Google Chrome o novou funkcionalitu. Vytváří tedy novou záložku, která reprezentuje přidanou funkcionalitu. Zachytává také události nad touto záložkou, např. její otevření. Po otevření takové záložky se zobrazí rozhraní nově definovaného nástroje, skrz které vývojář využívá jeho funkcí. Tato komponenta také vzniká dynamicky a každá zkoumaná webová stránka má svou vlastní instanci *Devtools page*. Nabízí nám rozhraní pro zjištění zdrojů webové stránky, či její síťové aktivity jejich jednotlivých částí. Jíná dostupná rozhraní nám zase umožní přímé vykonávaní Javascriptového kódu přímo na zkoumané stránce a také ověření jeho výsledku. V neposlední řadě také dokáže komunikovat s *Background page*, nikoli však s *Content scriptem*.

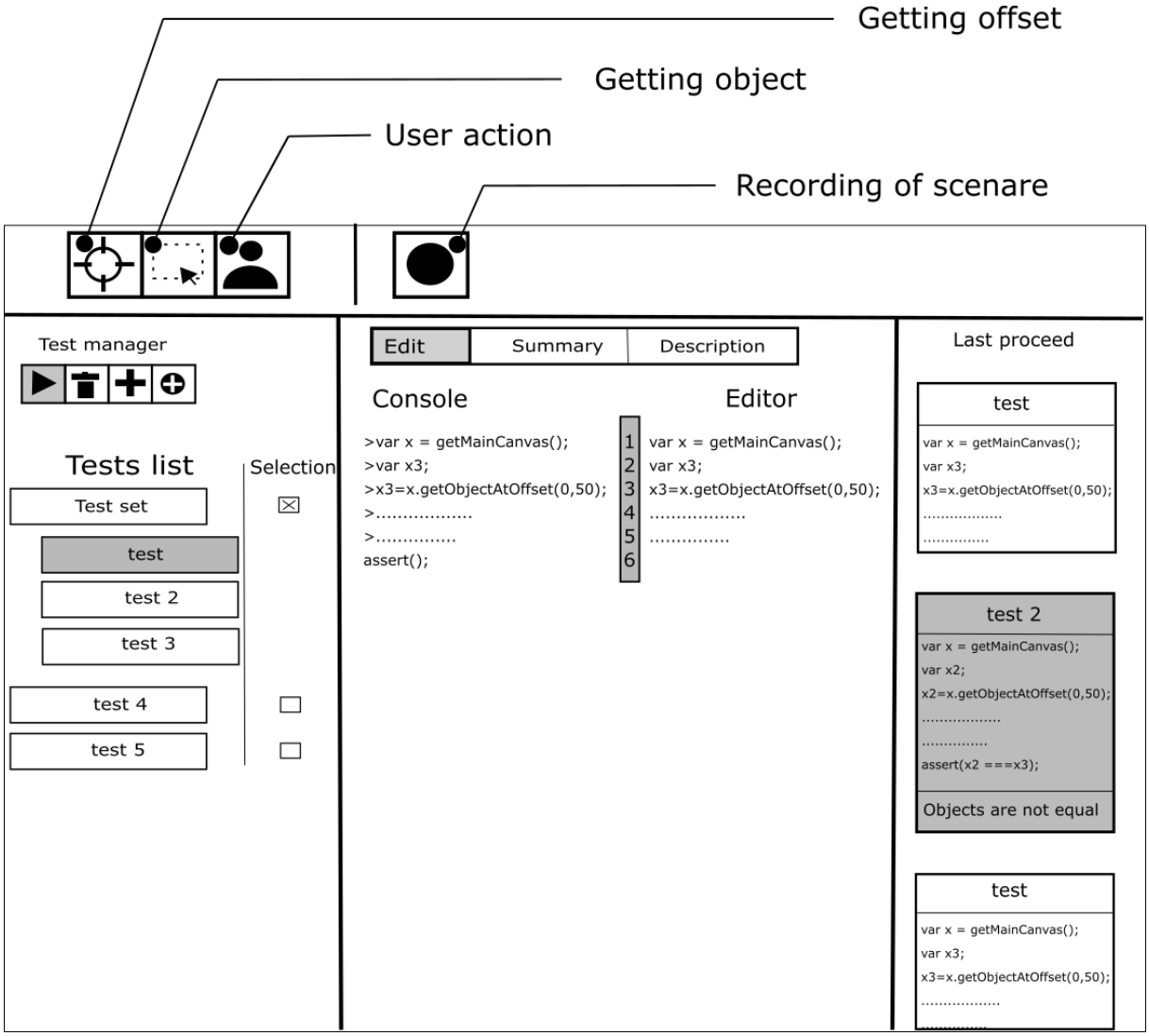
4.5 Návrh nástroje

Cílem této bakalářské práce je tvorba nástroje pro testování uživatelského rozhraní webových aplikací, a to ve formě doplňku pro prohlížeč Google Chrome. Každá část tohoto doplňku bude zajišťova rozličnou funkci. Základní jednotkou tohoto nástroje je testovací scénář. Z výše dostupných specifikací možností rozhraní jednotlivých komponent víme, že komponenta *Devtools page* umí provádět Javascriptový kód nad zkoumanou stránkou. Tohoto faktu využijeme a scénář bude sestávat z jednotlivých příkazů v Javascriptu. Tento kód ale bude muset využívat testovacího *API*, o kterém byla řeč výše. Abychom mohli univerzálně využívat tohoto *API*, je třeba vložit do testované stránky rozšiřující skript. Ten bude zajišťovat kontrolu nad testovacím *API* a přidávat funkce, umožnující interpretovat požadavky uživatelského rozhraní vyvíjeného nástroje do podoby příkazu ve scénáři, a také jeho inicializaci o referenci na primární objekt kreslícího plátna. Funkce pro manipulaci *DOM*, komponenty *Content script*, využijeme pro vložení rozšiřujícího skriptu do webové stránky. Abychom ale mohli využít *Content scriptu*, bude jej nejprve třeba dynamicky vložit do webové stránky, a to po vyvolání určité uživatelské události. Jelikož cílíme na vývojáře, vhodnou událostí by bylo otevření záložky vývojářského okna prohlížeče. Nejprve tedy bude potřeba pomocí komponenty *Devtools page* zachytit tuto událost a dále o ní notifikovat *Background page*, která zajistí samotné vložení *Content scriptu* do testované aplikace. Otevřená záložka v panelu vývojářských nástrojů bude představovat samotné uživatelské rozhraní testovací aplikace.

Toto rozhraní se bude skládat s několika oddělených částí. Hlavní menu bude interpretovat funkce testovacího *API*, jako např. výběr objektu kreslícího plátna. Dalé bychom skrze něj měli ovládat zaznamenání těchto akcí do scénáře, či mimo něj. Poté bychom si měli vytvořené scénáře ukládat a zobrazovat například v levém spodním sloupci, kde se mohou objevovat dva typy záznamů. Prvním typem je jeden samostatný scénář, druhým naopak skupina scénářů, schlukující do sebe několik scénářů. Nad tímto seznamem by mělo být levé menu, jež slouží pro správu testů. V tomto menu nalezneme tlačítka pro přidání testu samotného, skupiny testů, jejich smazání a spuštění. Při vytváření nového scénáře je třeba hledět, zdali jej nepřídáváme do aktuálně vyznačené skupiny. Pokud máme označený test, tak do něj už přidávat nelze, tudíž se vytvoří nová položka v seznamu scénářů a skupin.

V prostřední částí se zobrazí záložka pro editace, jež se bude sestávat z Javascriptové konzole v levé částí, v které bude moci vývojář provádět příkazy a interpretovat uživatelské akce v nástroji. Pokud bude spuštěno nahrávání, zaznamenají se tyto příkazy do editoru v pravé části, představující aktuální stav scénáře. Do tohoto editoru se bude moci dolpnit kód scénáře i ručně (kdekoli mezi příkazy). Pokud však bude mít uživatel označenou skupinu scénářů, v záložce pro editace se zobrazí čistě jen konzole. Scénáře se budou spouštět stiskem odpovídajícího tlačítka v levém menu a výsledek jejich průběhu bude zobrazen v pravém panelu. Pokud bude spuštěna skupina testů, zobrazí se v tomto panelu výsledky průběhy všech testů, spadajících do této skupiny. V další záložce shrnutí nalezneme všechny průběhy testů, které od sebe budou

rozlišeny pomocí časového údaje jejich spuštění. V případě skupiny nalezneme u každého testu jeho jednotlivé průběhy. V poslední záložce najdeme formuláře, kde lze zaznámenávat popisky jednotlivých testů, či celé skupiny. Skicu výše uváděného návrhu nalezneme na obrázku 7.



Obrázek 7: Wireframe *GUI* implementovaného rozšíření pro Google Chrome

5 Implementace

Podrobně si popíšeme samotnou architekturu testovacího nástroje a jeho komunikaci s webovou aplikací. Nakonec rozebereme jednotlivé komponenty.

5.1 Architektura

Nástroj je vyvíjen formou doplňku pro prohlížeč Google Chrome. Musí být tedy vytvářen na základě určitých standardů a specifikací, které tato forma vyžaduje. Struktura takové aplikace je pevně daná. Prvím velmi důležitým souborem doplňku je konfigurační manifest, což je soubor v *JSON* formátu. V tomto souboru najdeme základní vlastnosti našeho nástroje. Prvním parametrem je název rozšíření. Dále definujeme z jakých komponent se skládá, a poté každou z nich blíže specikujeme, např. k nim přidružíme jednotlivé skripty. Také definujeme jejich chování, např. *Background page* nastavením vlastnosti `persistent` na hodnotu `false` říkáme, že se bude chovat jako *Event page*. V neposlední řadě specifikujeme, jaké zdroje ve formě skriptů můžeme využívat. V sekci vyjímek také vyjmenováváme *API* platformy *Chrome*, kterých budeme v doplňku využívat. Implememtaci manifestu testovacího nástroje lze vidět na výpisu kódu.

```
{
  "manifest_version": 2,
  "name": "GUI Tester",
  "description": "This extension test GUI elements of web page",
  "version": "1.0",
  "background": {
    "scripts": ["background.js"],
    "persistent": false
  },
  "devtools_page": "devtools.html",
   "web_accessible_resources" : ["extension-script.js", "jquery.min.js"],
  "permissions": [
    "https://*/",
    "file://*/",
    "tabs",
    "activeTab",
    "https://ajax.googleapis.com/"
     ]
}
```

Dalšími částmi jsou již dříve zmíňovaný *Content script*, *Background page* a *Devtools page*. Každý modul implementuje odlišnou funkci skrze *API* platformy *Chrome*. Dále jej tvoří pomocné soubory jako rozšiřující skript a jQuery knihovna. *Content script* běží v kontextu tes-

tované webové aplikace a dovoluje přímý přístup k *DOM*. *Content script* poté dovoluje komunikaci s *Background page* skrze objekt `chrome.runtime`. *Background page* vkládá *Content script* pomocí `chrome.tabs`. *Devtools page* ovlivňuje testovanou webovou aplikaci pomocí metod `devtools.inspectedWindow`. Rozšiřující skript implementuje metody potřebné pro testování aplikace, např. asserty a dále metody vyvolávajících se na základě uživateslkých akcí v nástroji.

5.2 Struktura komunikace modulů

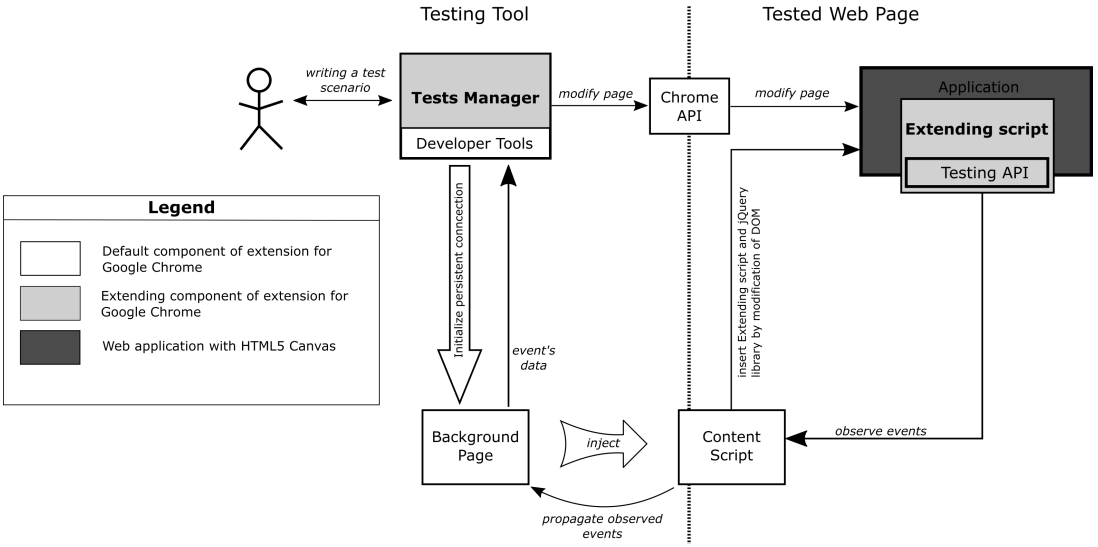
V rámci prohlížeče máme na straně jedné aplikaci, již budeme testovat a na straně druhé nástroj v podobě doplňku s pomocí nehož aplikaci testujeme. Mezi těmito aplikace musí probíhat komunikace. Jelikož se jedná o doplněk, jsme limitování jeho specifikacemi, které nám stanovují určitá omezení. Ná obrázku 8. vidíme podrobný diagram této komunikace nástroje s aplikací.

Vstupní událostí je otevření okna vývojářských nástrojů prohlížeče Google Chrome. V tomto okamžiku se vytvoří nová záložka *GUI* tester a dojde k zavolání metody `runtime.connect` v kompontě *Devtools page*, čímž dojde k vytvoření perzistentního spojení mezi touto komponentou a *Background page*. Parametrem této metody je název portu, který toto spojení jednoznačně identifikuje. Toto spojení odkazuje na nově vytvořenou záložku našeho nástroje. Po otevření a zobrazení této záložky se vyvolá událost `extensionPanel.onShown` *Devtools page*. Jejím parametrem je posluchač, ten nejprve pošle zprávu o inicializaci *Background page*, která po zachycení této zprávy vyvolá vložení *Content scriptu* skrze metodu `tabs.executeScript`. Jejím parametrem je cesta ke skriptu, jež pro nás představuje tuto komponentu. *Content script* dále pomocí `document.getObjectByTagName` nalezne element `head` webové stránky a následně na jeho konec nejprve vloží element `script` odkazující na jQuery knihovny a za něj další element `script`, nyní odkazující na rozšiřující skript.

Při stisku tlačítka pro získání offsetu uvnitř kreslícího plátna nebo pro výběr objektu v uživatelském rozhraní testovacího nástroje dojde k vyvolání odpovídající metody v rozšiřujícím skriptu skrze `inspectedWindow.eval`. Ta má za parametr Javascriptový kód, který se vykoná uvnitř testované stránky. Jakmile se provedou výpočty, je výsledek uložen do cachovací proměnné uvnitř webové stránky a z rozšiřujícího skriptu je vytvořena a následně vyvolána odpovídající událost. *Content skript* tuto událost skrze *DOM* zachytí spolu s jejím paramterem. Tuto událost propagujeme zaslaním asynchroní zprávy *Background page* na objekt `chrome.runtime`. Tento způsob komunikace nabízí z dostupných možností úsporu prostředků, jelikož jsou zprávy vypropagovány až nastanou, což je opak perzistentního spojení mezi *Devtools* a *Background page*, jež zabírá kapacitu jednoho spojení i když jím neprochází žádné zprávy. Při opětovném spuštění, či zadávání příkazů do konzole se opět využívá metody `inspectedWindow.eval`, ale navíc se provádí věci spojené se zaznamenáním výsledků do scénáře a kontrola výsledku dotazu, případně jeho zachycení.

Background page tedy využívá obojích metod komunikace pro zachycení události od *Content scriptu*. Naslouchá na asynchroní zprávy z objektu `chrome.runtime`. Pro propagaci těchto udá-

lostí do *Devtools page* využívá naopak perzistentního spojení na daný port, který představuje stále spojení s ním. Takovéto události tedy opět s jejími parametry odesílá skrze port. *Devtools page* zase na tomto portu naslouchá a zachytává tyto události. Z této komponenty se události již nikde dále nepropagují. *Devtools page* také obsahuje přímý odkaz na objekt `window` vývojářského nástroje. Na základě parametru vypropagované události se pouze vyvolá odpovídající metoda nad objektem `window` a testovací nástroj se dále postará o konečné zpracování a interpretování události do jeho prostředí. Příkladem je vykonání příkazu do konzole a jeho následné promítnutí do editoru. Kód volaný nad testovanou stránkou je trojí povahy. První skupinou je kód vyvolaný uživatelskými akcemi v nástroji, obsluhovaný rozšiřujícím skriptem a následně vypropagovaným do konzole ve správci testů. Druhou skupinou jsou příkazy ručně psané do konzole a poslední skupinou tvoří příkazy vyvolané při spuštění testů, ty mohou být také ručně vloženy do editoru.



Obrázek 8: Diagram komunikace mezi aplikace s nástrojem

5.2.1 Rozšiřující skript

Strukturu rozšiřujícího skriptu tvoří cachovácí proměnné a instance třídy `GuiTester` a také metoda, která vyvolá událost inicializace primárního objektu kreslícího plátna. Instance objektu `Guitester` obsahuje metody, které obsluhují volání *Devtools page* na testovanou webovou aplikaci. Prvními důležitými metodami této instance jsou asserty, umožnující při vykonávání jednotlivých scénářů ověřovat definované stavy aplikace. Dalšími metodami jsou události myši, připojené k elementu *HTML5 Canvas* webové stránky. Naslouchače těchto událostí už volají samotné metody testovacího rozhraní, výsledky těchto metod jsou cachovány do proměnných a dále jsou vyvolána odpovídající události, jež jsou dále propagovány do doplňku.

5.2.2 Content script

Hlavním objektem tohoto skriptu je instance třídy `ContentController`. Tato třída obsahuje metodu pro vkládání skriptů do webové stránky, neboli metody upravující *DOM*. Po vytvoření instance `ContentControlleru` je tato funkce několikrát volána. Jejím parametrem je *URL* řetězec příslušného skriptu. Prvním voláním se vkládá jQuery knihovna a následně rozšiřující skript. Poslední metodou instance je metoda jež nastavuje posluchač na události *DOM*, vyvolané na webové stránce. Tyto události zachytí pomocí posluchače nastaveného metodou `window.addEventListener`. Posluchač zpracuje tuto zachycenou zprávu a skrze `runtime.sendMessage` tuto zprávu zasílá *Background page*.

5.2.3 Background page

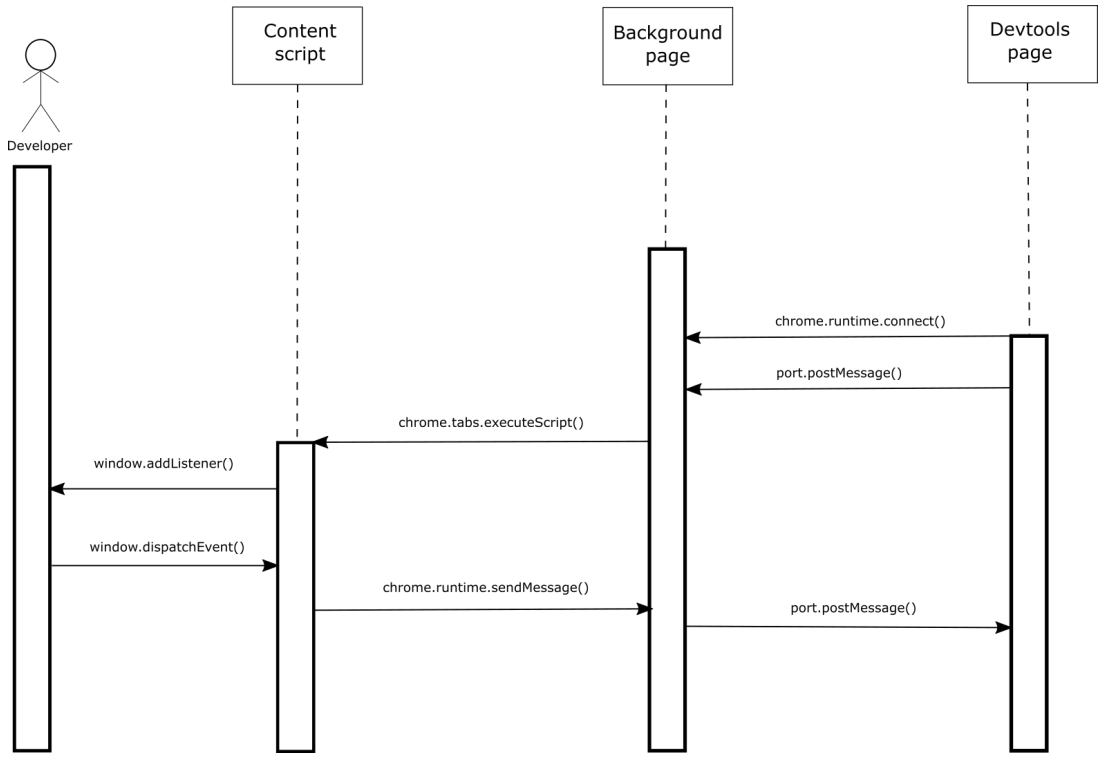
Background page tvoří most, který propojuje *Content script* s *Devtools page*. Jeho strukturu tvoří základní objekt, kterým je instance `BackPgControlleru`. Tento objekt zastřešuje chování celé této komponenty. První metodou, jež se volá po vytvořenní této instance je posluchač události `runtime.onConnect.addListener`, jejím parametrem je port, odkazující na panel ve vývojářských nástrojích prohlížeče Google Chrome. V rámci této metody se dále volá posluchač události `port.onMessage.addListener`. Parametrem tohoto posluchače je zachycená událost odchycená skrze perzistentí spojení od *Devtools page*. Tato událost je vypropagována, pokud dojde k otevření záložky testovacího nástroje v prohlížeči. Každé takové spojení označujeme jako port a ten je identifikován svým jménem. Proto musíme při komunikaci kontrolovat jejich označení a propagovat zprávy jen skrze odpovídající porty, jelikož může ve stejný okamžik být vytvořeno více portů. Z tohotu důvodu si uchovávame v instanci `BackPageControlleru` pole schválených portů. Pro obsluhu tohoto pole slouží metody jako `pushPort`, `indexPort` apod. Poté se v této metodě volá funkce `tabs.executeScript`, sloužící pro vložení skriptu do webové stránky. Jejím parametrem je odkaz na vkládaný skript.

Poslední funkcí této komponenty je zachytávaní zpráv *Content scriptu* a následně jejich přeposílání dále *Devtools page*. Směrem od *Content scriptu* jsou zachytáváný posluchačem na objektu `chrome.runtime.onMessage`. Tyto zprávy jsou propagovány asynchroně a jakmile jsou zachyceny, prochází se pole uložených portů instance `BackPgController` a na jednotlivé porty je volána událost `port.sendMessage`, zasílající tímto perzistentním spojením komponentě *Devtools page* dříve zachycenou událost.

5.2.4 Devtools page

Devtools page tvoří poslední komponentu našeho rozšíření, která představuje nejpodstatnější část. Znamená pomyslný okraj doplňku a také poslední komunikační uzel. Jeho jediný objekt je instancí třídy `DevtoolsController`. Instanční proměnné této třídy tvoří odkaz na okno panelu vývojářských nástrojů prohlížeče, představující samotné uživatelské rozhraní nástroje pro spravování testů, dále název portu, odkaz na tento port a nakonec odkaz na samotný panel ve

vývojářských nástrojích. Při jeho otevření dojde k vytvoření instance `DevtoolsControlleru` a v rámci ní se volá metoda `panel.create`. Jejími parametry jsou název nově vytvořeného panelu a webová stránka zobrazující se uvnitř něj po jeho otevření. Posledním parametrem je funkce zajišťující chování tohoto panelu. V této funkci dojde k inicializaci perzistentního spojení pomocí `chrome.runtime.connect`. Dalšími krokem této metody je nastavení posluchače při odchycení události `ExtensionPanel.onShown`. Tento posluchač nejprve na vytvořený port zašle událost, která notifikuje *Background page* o otevření panelu a uloží odkaz na objekt `window`, představující stránku správce testů. Také dojde k nastavení posluchače na zprávy zaslané *Background page* pomocí `port.onMessage.addListener`, tento posluchač neprve přečte detail zprávy a na jeho základě volá danou funkci na objekt `window` správce testů. Sekvenční diagram zobrazuje na obrázku 9. komunikaci v čase mezi jednotlivými komponentami.



Obrázek 9: Diagram komunikace mezi komponentami doplňku

5.2.5 Správce testů

Správce testů představuje stránku, zobrazující se uvnitř panelu vývojářských nástrojů. Jedná se o nesložitější část vytvořeného testovacího nástroje. Využívá několika rozšíření, vytvořených pro webové aplikace. Dvěma základními rozšířeními jsou jQuery knihovna a framework Bootstrap, dále využívá pluginu jQuery Terminal Emulator [12], který vytváří interpret příka-

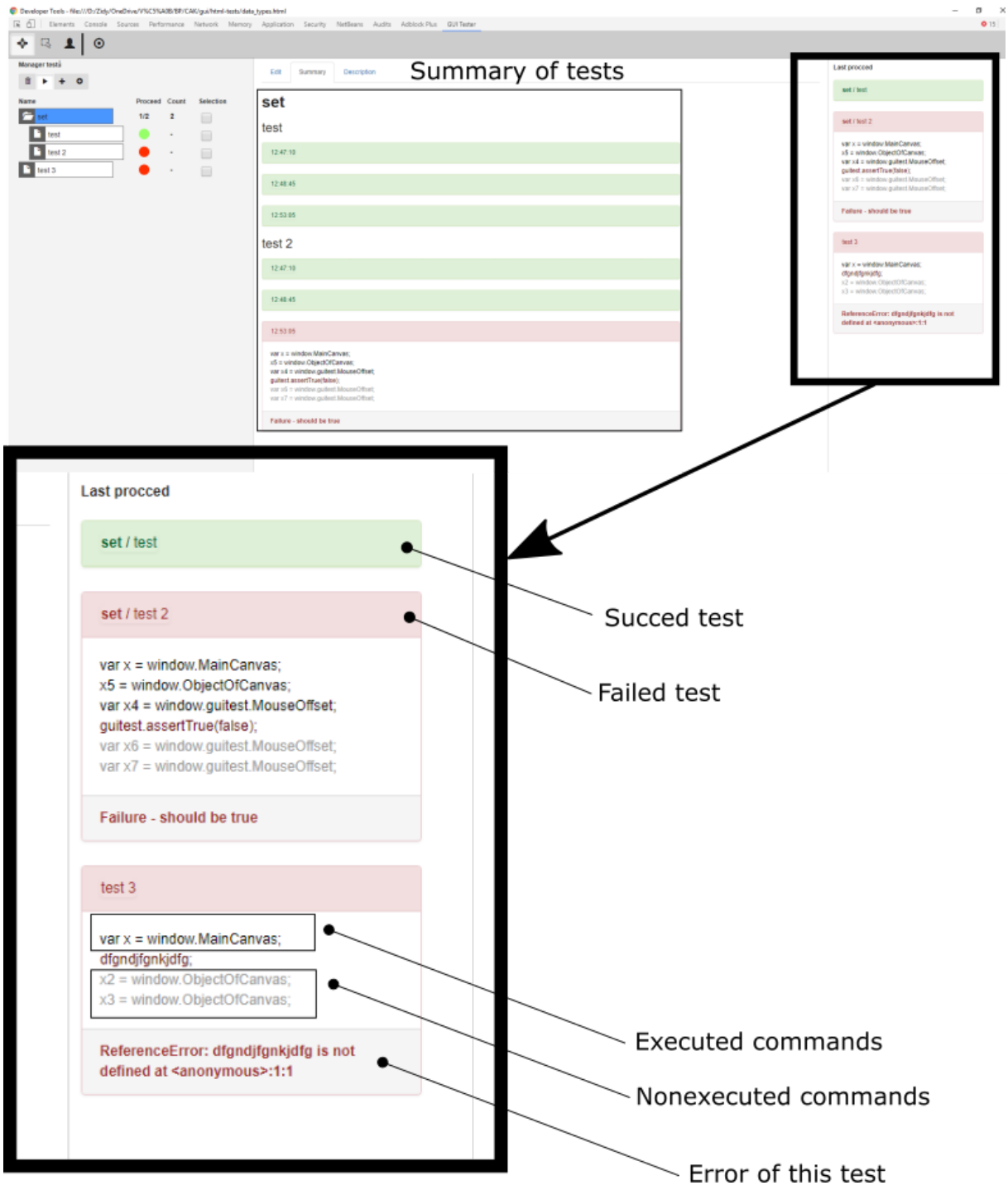
[12] `http://www.terminal.jcubic.pl/`

zové řádky jazyka *Javascript*. Dalším a posledním rozšířením je ACE [13], což je editor kódu pro různé jazyky. V našem případě se jedná o jazyk *Javascript*. Struktura této komponenty tvoří aplikaci, založenou na modelu *MVC*. Komponenta model tedy představuje datovou vrstvu, ktetá se skládá z tříd `Model`, `ScenareGroup`, `Scenare`, `Run` a `Error`. Třída `Model` obsahuje referenci na primární objekt kreslícího plátna, dále čítač pro získávání identifikátoru skupiny nebo scénáře a posledními atributy jsou pole obsahující scénáře nebo skupiny scénářů a atribut uchovávající si aktuálně vybraný prvek tohoto pole. Třída `ScenareGroup` slouží ke shlukování scénářů. Třída `Scenare` tedy představuje základní scénář a jejími atributy jsou unikátní identikátor, vygenerováný třídou `Model`, název, jeho jednotlivé příkazy a nakonec pole představující jednotlivá spuštění. Každé takové spuštění poté obsahuje atribut jeho výsledku, a krom pole vykonávaných příkazů také atribut, obsahující vzniklou chybu. Třída `Model` již dále obsahuje jen metody pro práci se scénáři, které slouži pro jejich uložení, získání a úpravu.

Komponenta *View* má jedinou třídu stejnojmeného názvu, která se stará o zobrazení a vykreslení jednotlivých prvků uživatelského rozhraní, vytvořeného testovacího nástroj. Hlavním atributem je instanci třídy `Model`, uchovávající data, jež jsou dále instancí třídy `View` zobrazována v layoutu stránky. Ostatními atributy jsou JQUERY objekty, odkazující na jednotlivá menu, panely a další důležité elementy stránky. Tyto atributy mohou obsahovat jak skupinu tlačítek, tak jediné vstupní pole formuláře. Dále tato třída obsahuje metody vykreslující seznamy scénářů a skupin scénářů a vkládá formuláře pro jejich přidání. Ostatní metody se starájí o vykreslení kontextu uprostřed stránky, což může být např. výpis dokumentace testů, nebo přehled jednotlivých spuštění těchto testů. Poslední jsou metody, zajišťující vykreslení testů v panelu, zobrazujícím poslední spuštěné testy.

Controller představuje poslední komponentu, starájící se o funkčnost rozšíření, kterých správce testů využívá. Jedná se konkrétně o konzoli a editor jazyka *Javascript*. V případě konzole implementuje chování funkce, obsluhující vykonání příkazů z konzole a při ukládání scénářů získává aktuální obsah konzole a také editoru kódu. Funkce vykonání dotazu využívá *API* platformy *Google* a to kokrétně funkce `devtools.inspectedWindow.eval`, umožňující vykonání Javascriptového kódu na testované stránce. Při výběru objektu v testované aplikaci zase zajišťuje interpretaci této akce v podobě příkazu do konzole a dále vložení tohoto příkazu na nový řádek v editoru kódu, pokud je nastaven příznak pro nahrávání scénáře. Atributy, které obsahuje, tvoří instance třídy `View` a `Model`, dále příznak pro nahravání a provádění scénáře. Dalšími funkcemi tato komponenta obsluhuje chování testovacího nástroje na uživatelské podněty, příkladem můžeme uvést změnu aktuálního scénáře kliknutím na jiný scénář v jejich přehledu. Při této události musí instance `Controlleru` nejprve uložit změny provedné v aktuálním scénáři do instance `Modelu`. V dalším kroku musí naopak načíst nově vybraný scénář a voláním odpovídajících metod označit nově vybraný scénář v jejich přehledu a dále zkontrolovat, jaký kontext je vybrán (editace, přehled, dokumentace). Podle něj musí vyvolat odpovídající metodu `View`. Zakladní kostru správce aplikací tvoří *HTML* stránka. Na obrázku 10. vidíme podobu nástroje pro testování aplikací.

[13] `https://ace.c9.io`



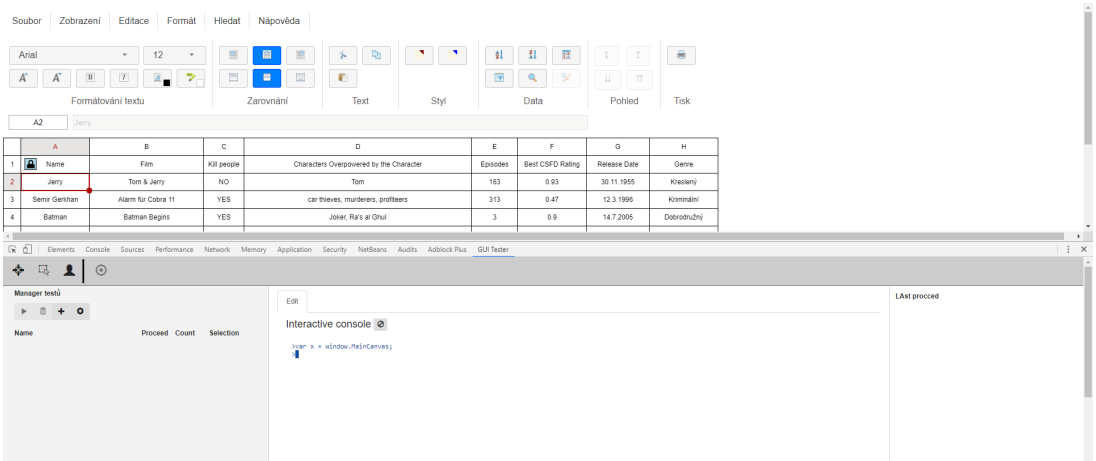
Obrázek 10: Rozhraní nástroje pro testování

6 Používání nástroje

V této kapitole si ukážeme ukázkový příklad práce s tímto testovacím nástrojem. Popíšeme si jak vytvářet a spravovat testy. Nakonec si ukážeme samotné spuštění testů.

6.1 Tvorba testů

Nyní se nacházíme na začátku a jsme v tzv. výchozím bodě. Máme před sebou otevřen webový prohlížeč Google Chrome a v jeho aktuální záložce máme otevřenou webovou aplikace, kterou budeme chtít testovat. Dále se je třeba dostat do panelu vývojářských nástrojů stiskem kombinace kláves CTRL-SHIFT-I. Na tomto panelu se nám zobrazí několik záložek(elementy, konzole, zdroje atd.), z kterých je třeba kliknout na tu, jež má název *GUI* tester. Po tomto úkonu se zobrazí nástroj pro testování ve výchozím stavu. Tento stav odpovídá konzoli obsahující jediný příkaz `x = window.MainCanvas`, tlačítko pro nahrávání do scénáře je v neaktivním stavu a na kontextovém panelu je viditelná pouze záložka Edit. Výchozí stav aplikace prezentuje obrázek 11. V případě špatně vytvořeného scénáře je možné jej vymazat zaškrnutím checkboxů jednotlivých položek a stisknutím tlačítka pro jejich vymazání umístěného v levém menu.



Obrázek 11: Inicializace nástroje pro testování

Nyní můžeme přejít k dalšímu kroku, ve kterém budeme vytvářet samotné testovací scénáře. Nejprve si pomocí tlačítek v levém menu vytvoříme strukturu scénářů. Pokud přidáváme nový scénář a máme aktuálně označenou skupinu scénáru, nový scénář se stává její součástí, pokud je aktuálně vybrán jen nový scénář, vytvoří se jako samostatná položka. Testovací skupina je vždy vytvořena jako samostatná položka, kterou poté plníme scénáři. Jakmile máme scénářovou strukturu, můžeme jetnotlivé scénáře editovat. Příklad scénáře můžeme vidět na tomto seznamu.

1. referenční objekt plátna je automatickou součástí scénáře
2. získáme první objekt plátna kliknutím myší do kreslícího plátna a zároveň zakliknutím tlačítka v nástroji pro testování, jež této funkci odpovídá

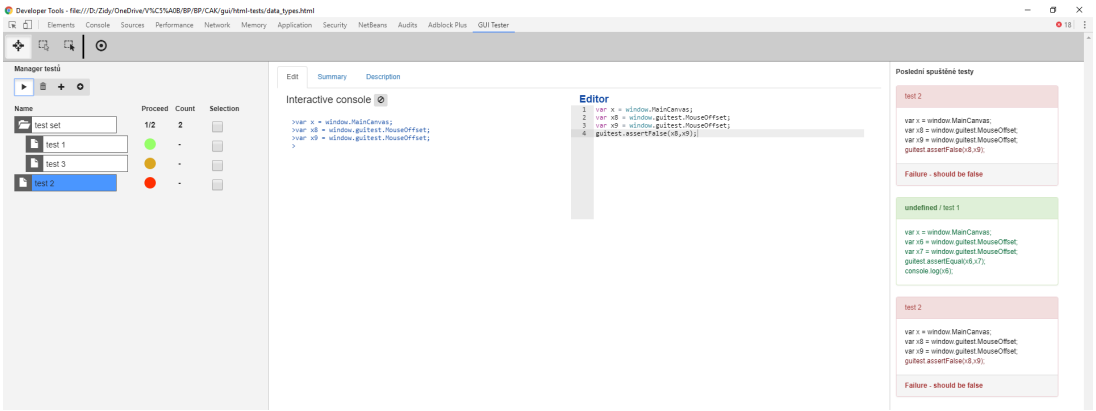
3. získáme offset `MouseOffset()` kliknutím myší do kreslícího plátna, a zakliknutím tlačítkem v nástroji pro testování, jež této funkci odpovídá

4. zavoláme metodu `getObjectAtOffset()` na primární referenci, jejíž parametry budou souřadnice získaného offsetu a získáme druhý objekt

5. vytvoříme `assertEqual()` a parametry budou oba získané objekty

6.2 Dokumentace a spouštění testů

Před spuštěním testů je možné zaznamenat poznámky k jeho dokumentaci pomocí kontextové záložky Description a stisknutím tlačítka pro záznam této dokumentace pod formulářem. Poté lze zakliknutím jednotlivých checkboxu označit ty testy, které budeme chtít spustit. Poté je potřebat stisknout příslušné tlačítko pro spuštění scénářů a v pravém panelu se nám zobrazí aktuální výsledky jednotlivých průběhů. V kontextové záložce Summary poté vydíme záznamy jednotlivých průběhů daného test. Přehled vidíme na obrázku 12.



Obrázek 12: Spuštěné testy

7 Závěr

Cílem práce bylo zhotovit komplexní nástroj pro testování uživatelského rozhraní. Výsledkem je tedy nástroj ve formě doplňku prohlížeče Google Chrome, umožnující tvorbu testovacích scénářů ve formě posloupnosti příkazů, které je možné uložit a posléze opětovně spustit. V průběhu tvorby jsem se setkal s několika problémy, které se vždy nakonec podařilo vyřešit nastudováním dané problematiky více do hloubky nebo pomocí alternativního řešení. Jako příklad můžeme uvést komunikaci mezi částí *Content script* a *Devtools page* nebo implemetace konzole pro tvorbu scénářů, jež jsem vyřešil použitím pluginu jQuery Terminal Emulator. Ukázali jsme si příklad otestování *RIA* aplikace pro plánování lidských zdrojů, na kterém jsme si ověřili funkčnost tohoto řešení. Celkové řešení tedy splňuje zadaní a umožnuje tvorbu testovacích scénářů a jejich hromadné spouštění, přičemž v případě neúspěšných testů uživateli nabídne sekvenci příkazů, vedoucí k této chybě.

Vytvořené řešení lze využívat univerzálně v jakékoli jiné webové aplikaci, obsahující prvek *HTML5 Canvas*. Jedinou podmínkou je implemetace testovacího *API*. Tento nástroj tedy představuje existující alternativu k nástrojům jako Selenium, či Screenster, ale oproti nim nabízí možnost úplného testovaní *RIA* webových aplikací. Tento fakt jej činí konkurenceschopný.

Nástroj v aktuální podobě umožní přenostitelnost testovacího scénáře jeho zkopirováním z editoru kódu a uložením do textového souboru. Jedním z budocích cílů rozšíření tohoto nástroje je export testovacích scénářů přímo na uložiště GitHub, což by představovalo velkou výhodou v případě týmového vývoje. Závěrem bych chtěl zmínit omezení tohoto nástroje, tím je orientace pouze na jednu platformu prohlížeče, konrétně Google Chrome, jelikož jsem zjistil že jednotlivá *API* nejsou mezi prohlížečem Google Chrome a Chromium kompatibilní.

Literatura

[1] Alan Page, B. R., Ken Johnston: *Jak testuje software Microsoft*. Brno: Computer Press, první vyd, 2009, ISBN 978-80-251-2869-5.

[2] Glenford J. Myers, T. B., Corey Sandler: *The Art of Software Testing*. New Jersey: Tom Badgett and Todd M.Thomas with Corey Sandler, druhé vyd, 2004, ISBN 0-471-46912-2.

[3] jQuery, e. k.: *jQuery-Kuchařka programátora*. Brno: Computer Press, Albatros Media a.s., první vyd, 2010, ISBN 978-8-025-13952-3.

[4] Škultéty, R.: *JavaScript: programujeme internetoveé aplikace*. Brno: Computer Press, první vyd, 2004, ISBN 80-251-0144-4.

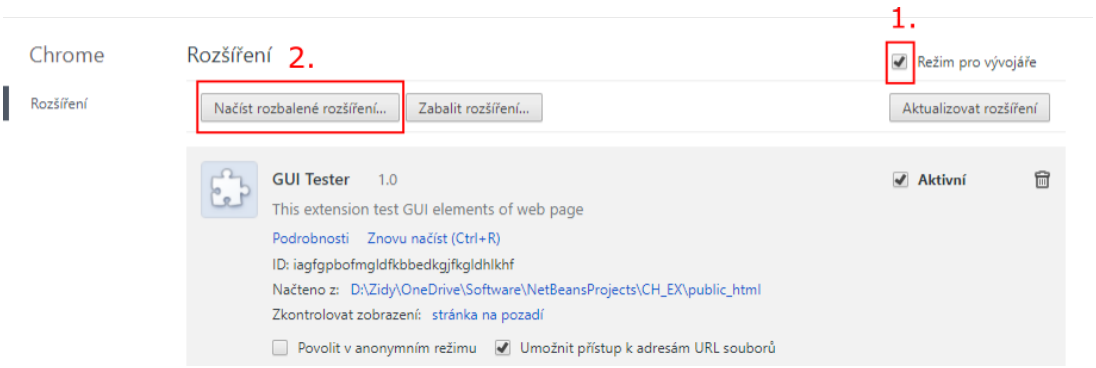
[5] Patton, R.: *Testování softwaru*. Praha: Computer Press, první vyd, 2002, ISBN 80-7226-636-5.

[6] Rudolf, P.: *Návrhové vzory*. Brno: Computer Press, první vyd, 2013, ISBN 978-80-251-1582-4.

A Instalace

Nejprve je třeba naklonovat zdrojové kódy testovacího nástroje na z uložiště GITHUB [14]. Poté je třeba v kořenové složce staženého nástroje naklonovat modul ACE, opět ze stejného uložiště [15]. Další postup tkví ve vykonání kroků, jdoucích k integraci testovacího nástroje do prohlížeče. Tyto kroky jsou shrnuty v následujícím seznamu. Po této integraci do prohlížeče může započít samotné testování touto aplikací.

1. Nainstalujeme prohlížeč GOOGLE CHROME.
2. V sekci rozšíření na obrázku 13., která se nachází v nastavení prohlížeče zaškrtneme režim pro vývojáře.
3. Poté stiskneme tlačítko Načíst rozbalená rozšíření.
4. Jako zdrojovou složku vybereme složku, která obsahuje testovací nástroj.
5. Nyní je tento nástroj v podobě doplňku integrován do prohlížeče.



Obrázek 13: Nahrání rozšíření

A.1 Řešení známých problémů

Pokud se při spuštění nástroje neobjeví základní příkaz v konzoli, je třeba v sekci prohlížeče aktualizovat rozšíření a poté se ještě musí aktualizovat i testovaná stránka. Při následném spuštění by mělo dojít ke správné inicializaci nástroje.

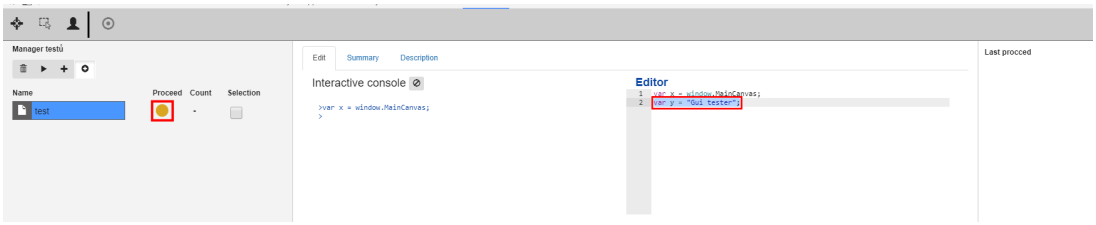
[14] `https://github.com/ZidyGT/GUI-Tester.git`
[15] `https://github.com/ajaxorg/ace-builds.git`

B Příloha na CD

V přílohách nalezneme *CD*, které obsahuje zdrojovou aplikaci a přiložené testy pro ověření funkčnosti. Kořenový adresář *CD* obsahuje 2 složky. Ve složce Tool nalezneme zdrojové kódy testovacího nástroje. Ve složce Scenarios nalezneme příklady testovacích případů v textové podobě.

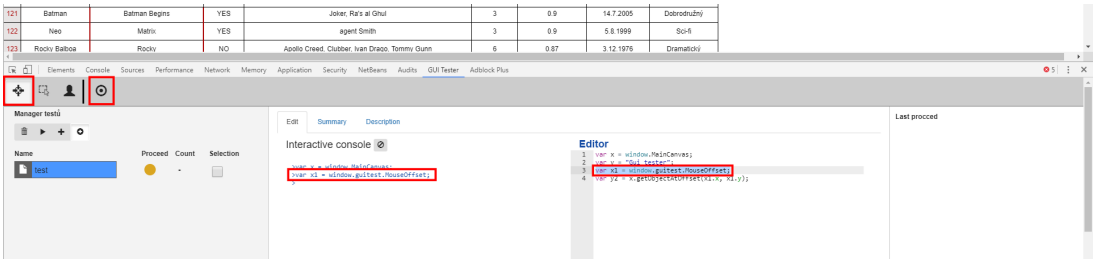
B.1 Příklad prvního testu

V kapitole použití jsme si uvedi jak si vytvořit test. Máme tedy před sebou nově vytvořený test a začneme vytvářet jeho strukturu. První příkaz už v něm tedy máme. Nyní dopíšeme do scénáře příkaz `var y = "Gui tester";` (obrázek 14.). Posledním krokem je ověření stavu aplikace pomocí příkazu `guitest.assertEqual(y, y2.value);`.



Obrázek 14: Základ scénáře

Dalším naším krokem bude získání offsetu objektu uvnitř kreslícího plátna. Nejprve je třeba kliknout na ikonu nahrávání a poté na tlačítko označující vybírání offsetu (obrázek 15.).

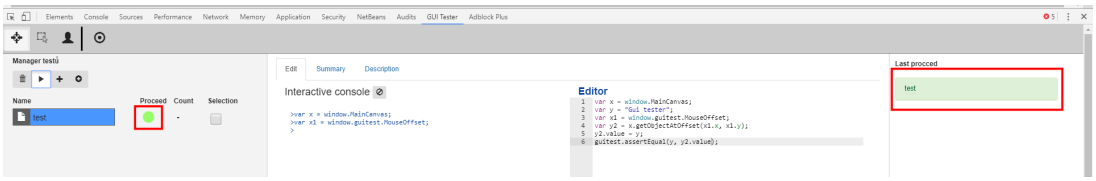


Obrázek 15: První krok ve scénáři

Nyní je aplikace správně nastavena a je možné v testované aplikaci vybrat offset kreslícího plátna. Poté napíšeme do scénáře příkaz `var y2 = x.getObjectAtOffset(x1.x, x1.y);` a `y2.value = y;` (obrázek 16.).

Obrázek 16: Spuštění scénáře

Poté zaškrtneme políčko ve sloupci Select odpovídajícího scénáře. Posledním krokem je kliknutí na tlačítko spuštění testu v levém menu. Výsledek vidíme na obrázku 17.



Obrázek 17: Výsledek scénáře